DTIC

AD-A226 495

MENTATION PAGE

Form Approved
OPM No. 0704-0188

to average 1 hour per response, including the time for reviewing instructions, searching existing data sources gathering and ... collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions ... to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Information and Regulatory Affairs, Office of Management and Budget, Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	27 August 90	Working Draft

4. TITLE AND SUBTITLE

Ada 9X Project Report: Ada 9X Requirements Document

5. FUNDING NUMBERS

6. AUTHOR(S)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Ada Joint Program Office
The Pentagon, Rm. 3E114
Washington, D.C. 20301-3080

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Ada Joint Program Office
The Pentagon, Rm. 3E114
Washington, D.C. 20301-3080

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited distribution

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document contains a distillation of requests for language changes submitted by the general public and from special workshops held to identify potential areas for revision. The purpose of this document is to specify needs that are considered to be the appropriate focus of the Ada 9X revision effort and to identify revision requirements that are to be satisfied by the Mapping/Revision Team.

Keywords: ... parallel ... distributed ... (IRC)

14. SUBJECT TERMS	15. NUMBER OF PAGES
Ada 9X, requirements	48
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to **stay within the lines to meet optical scanning requirements.**

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD - See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE - See authorities.
NASA - See Handbook NHB 2200.2.
NTIS - Leave blank.

Block 12b. Distribution Code.

DOD - DOD - Leave blank.
DOE - DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA - NASA - Leave blank.
NTIS - NTIS - Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

Ada 9X Project Report

Draft
Ada 9X Requirements Document

August 1990

Office of the Under Secretary of Defense for Acquisition

Washington, D.C. 20301

Table of Contents

Preface

This document is a *Working Draft* of the Ada 9X requirements. I am releasing this early version in order to keep you informed about the Project's status. Please note this document represents work in progress. It conveys the general direction being taken but is by no means complete. Your comments are welcome and should be sent to the following address by 1 October 1990.

The Ada 9X Requirements Team
c/o Dr. John Goodenough
Software Engineering Institute
4500 Fifth Avenue
Carnegie Mellon University
Pittsburgh, PA 15213

FAX: (412) 268-5758

E-Mail: **ada-comment@ajpo.sei.cmu.edu**

Comments should use the following format:

!section ss.ss(pp) Commenter Name yy-mm-dd
!version 3.3
!topic Title summarizing comment

where ss.ss is the section number of the document, pp is the paragraph number, and yy-mm-dd is the date the comment was composed. When making suggestions for improving the wording, please use square brackets [ ] to indicate text to be omitted and curly braces { } to indicate text to be added.

If possible, please send comments by e-mail. If your comments are lengthy and you cannot send them by e-mail, it would be appreciated if you could include a copy in machine-readable form, using either a Macintosh or IBM-compatible disk format.

The final Ada 9X Requirements Document will be released by 5 December 1990. There will also be a presentation on the requirements at the Tri-Ada Conference.

I appreciate your interest in the Ada 9X Project and look forward to your continued support in the remaining critical phases.

Christine M. Anderson
Ada 9X Project Manager

Foreword

The revision of ANSI/MIL-STD-1815A, the Ada programming language, is being conducted by the Ada 9X Project Office located at the Air Force Armament Laboratory at Eglin AFB, Florida. The Ada 9X Project is sponsored by the Ada Joint Program Office in the Office of the Under Secretary of Defense for Acquisition. The Project was initiated in October 1988 with an invitation to the public to submit revision requests. Over 750 revision requests were submitted worldwide between October 1988 and October 1989. In addition, several international workshops and meetings were organized by the Ada 9X Project Office for the purpose of further refining and prioritizing user needs in order that they might better be addressed in the revision effort. 1

There are many contributors to this Working Draft that deserve recognition: the Requirements Definition Team based at the Software Engineering Institute (SEI) which includes John Goodenough (SEI), Art Evans, Jr. (SEI), Ben Brosgol (Alsys), and Robert Dewar (NYU); the Requirements Analysis Team based at the Institute for Defense Analyses (IDA) which includes Cy Ardoin (IDA), Paul Baker (CTA), Douglas Dunlop (Intermetrics), Audrey Hook (IDA), Joseph Linn (IDA), Catherine McDonald (IDA), Reginald Meeson, Jr. (IDA), Steven Michell (Prior Data Sciences), and Karl Nyberg (Grebyn Corp); the Ada 9X Distinguished Reviewers which include E. Ploedereder — DR Chair (Tartan), G. Booch (Rational), B. Brosgol (Alsys), N. Cohen (IBM), R. Dewar (NYU), G. Dismukes (TeleSoft), D. Fisher (Incremental Systems), A. Gargaro (Computer Sciences), M. Gerhardt (ESL), J. Goodenough (SEI), E. Guerrieri (SofTech), *T. Harrison (SPC), S. Heilbrunner* (IABG — Germany), M. Henne (Sverdrup), P. Hilfinger (UC/Berkeley), Jean Ichbiah (Alsys — DR emeritus), M. Kamrad II (UNISYS), R. Landwehr (CCI — Germany), C. Lester (Portsmouth Polytechnic — UK), R. Leavitt (Prior Data — Canada), L. Månsson (Telelogic AB — Sweden), R. Mathis (Contel), M. Mills (US Air Force), C. Mitchell (DEC), D. Pogge (US Navy), K. Power (Boeing), O. Roubine (Consultant — France), T. Taft (Intermetrics), W. Taylor (Ferranti — UK), E. Vasilescu (GNV), and T. Wheeler (US Army); and the members of the Ada community who have so generously given of their time to attend workshops and special meetings, submit revision requests, and review and comment on interim reports. 2

On behalf of the Ada 9X Project Office, I wish to express sincerest thanks to all. 3

Christine M. Anderson
Ada 9X Project Manager

1 Introduction

This document contains a distillation of requests for language changes submitted by the general public and from special workshops held to identify potential areas for revision. The purpose of this document is to specify needs that are considered to be the appropriate focus of the Ada 9X revision effort and to identify revision requirements that are to be satisfied by the Mapping/ Revision Team.

1.1 Background

This document is the result of a collective effort to identify the most appropriate kinds of revisions that should be made to the Ada Programming Language (ANSI/MIL-STD-1815A-1983 and ISO 8652-1987). The effort began in January 1988 when the Ada Board was asked to prepare a recommendation to the Ada Joint Program Office (AJPO) on the most appropriate standardization process to use in developing a revised Ada standard, known as Ada 9X. The recommendation [2] was delivered in September 1988 to Virginia Castor, Director, Ada Joint Program Office, who subsequently established the Ada 9X Project for conducting the revision of the Ada Standards. Christine M. Anderson was appointed Project Manager in October 1988.

The Ada 9X effort consists of several phases. The output of the Requirements Definition Phase is this document, which specifies the revision needs to be addressed. The next phase is performed by the so-called Mapping/Revision Team, which is responsible for defining changes to the standard that meet these requirements. The purpose of the Requirements Definition Phase is to identify and prioritize the revision requirements to be satisfied by the Mapping/Revision Team.

The requirements definition effort and the mapping/revision efforts are both constrained by the overall objective of the Ada 9X effort [1]:

> Revise ANSI/MIL-STD-1815A-1983 to reflect current essential requirements with minimum negative impact and maximum positive impact to the Ada community.

A revision was undertaken because, according to the Ada Board, "some omissions, limitations, and minor errors have been discovered ... since the ANSI standard was approved. Although a major revision does not appear to be necessary to correct these problems, some revision is warranted." [2] The Ada Board further recommended that the scope of the revision be defined in a document that captures the high level requirements to be satisfied by the revision. The Board then went on to say, "The objective of the proposed revision should be to select only those changes that improve the usability of the language while minimizing the disruptive effects of changing the standard."

1.2 Scope of the Revision

1 Analysis of the revision needs expressed in the Revision Requests and User Forums suggests that different user communities have different needs, and that changes to Ada that are important to some application domains are either unnecessary or are viewed as detrimental to other application domains. For example, support for decimal arithmetic is important if Ada is to satisfy the needs of information systems applications, but real-time and embedded-system applications typically have no need for such numeric representations and would prefer that compiler vendors devote time and effort to supporting other specialized capabilities (often ones that are not critical to information systems). Educators are particularly interested in ensuring that the language can be taught easily (e.g., that there are simple rules with few exceptions) and that implementations are affordable and perform efficiently when compiling many small student programs, but these requirements are of less concern to other classes of users.

2 Implementers are concerned that unless the scope of the changes is small, the job of modifying compilers will be a major and expensive effort. Users are worried that if implementer resources are overextended, Ada 9X compilers will be less efficient, less reliable, and more costly than existing compilers (at least initially). Some users are also worried that if Ada is not enhanced in some areas (for example, by providing support for distributed systems, object-oriented programming, or information systems), it will fail to attract enough new users to ensure the continued improvement and availability of compilers.

3 Finally, a substantial minority of the revision requests submitted by the public are satisfied by some, but not all, implementations (usually in non-standard ways permitted by the language, e.g., special pragmas or compiler switches). For such requests, the language already allows the need to be satisfied but does not ensure that it *will* be satisfied. Changes that lead to more implementation uniformity would be helpful here, but increased uniformity is somewhat in conflict with the diversity of user needs as noted in the first paragraph.

4 After giving due consideration to the general nature of the requests for language changes, we have reaffirmed the essence of the Ada Board's conclusion that although improvements to Ada will be helpful in fostering its continued use, there are no fatal problems that prevent its successful use for many applications. Therefore, radical change is unjustified, and indeed, could have a significant negative impact on the ability of implementers to support user needs.

5 Equally important, after giving due consideration to the conflicting concerns expressed by members of the Ada community, we have established the following premise affecting the scope of the revision effort, the requirements specified in this document, and the structure of the resulting standard:

6 *It should be possible for an Ada 9X implementation to address just the needs of a selected group of users, e.g., hard real-time applications developers, information systems developers, educators, etc.*

7 This means the Ada 9X standard should be allowed to specify capabilities or implementation

constraints that are not appropriate for every application community and that therefore, need not be supported by every implementation. Such capabilities and constraints could be given in special annexes to the standard. The idea is that these annexes are part of the standard but need not be supported by every implementation.

Allowing specialized annexes should give greater freedom to the Mapping/Revision Team to address the needs of specialized application areas without having to be overly concerned that a specified capability is inappropriate for some community of users (since not every implementation will have to support the required capability). This represents a change from the existing policy that all implementations support the entire language.

Since inefficient support for certain uses of language constructs is tantamount to non-support as far as some applications are concerned, it is expected that each annex will address critical performance and support issues that cannot or need not be satisfied by all implementations. For example, a hard real-time annex might specify support for "immediate" abort, while an information systems annex might specify support for packed decimal arithmetic.

One advantage of this approach is that it allows for incremental transition to a full Ada 9X implementation — implementers can decide to concentrate initially on supporting just those annexes of greatest interest to their customer base. The other advantage, of course, is that it allows the specialized and critical needs of different user communities to be addressed explicitly in the standard.

The role of the annexes in compiler validation needs to be kept in mind during the revision. The Mapping/Revision Team should assume that validation policy can be developed to help ensure that the purpose of the specialized annexes is achieved. One possibility is that validation certificates would mention only those annexes that are fully supported by an implementation. Moreover, it is probably reasonable to allow an implementation to provide only partial support for additional annexes (without claiming validation status for such support). In such cases, an implementation probably should be required to warn users if they use a capability that goes beyond what is covered by a validation certificate. (Of course, care should be taken to avoid a useless proliferation of warning messages.) The details need to be worked out during the revision, and depend to some extent on the nature of the specialized annexes.

1.3 Structure and Terminology

This document is organized to reflect identified groups of needs for improving the usability of Ada. Each chapter is devoted to a different group of needs. After specifying a revision need in user-oriented terms, we specify language-oriented *Requirements* that are to be satisfied by the Mapping/Revision Team. Some requirements may be satisfied by providing additional language features (syntax, semantics, attributes, or pragmas) while others may simply impose performance constraints on implementations. Having the revised standard address performance issues reflects the fact that the *way* in which a feature is supported is sometimes critical to its usability in

particular applications. (For example, some aspects of run-time performance are critical for embedded real-time systems, e.g., real-time applications require that the run-time executive mask interrupts for a maximum (and known) amount of time.) It is expected that performance issues of particular interest to different user communities will be addressed in separate annexes of the standard.

2 We generally do not specify whether the solution to a particular requirement should be be given in one of the special annexes or in the main body of the standard. This decision is left to the Mapping/Revision Team because it depends on the nature of the solutions to the requirements. On the other hand, some requirements seem sufficiently specialized that they will probably addressed in a special annex; we sometimes mention this in our discussion of a requirement to remind the reader that we realize the requirement could lead to solutions that are of little interest or are detrimental for programming some applications.

3 A User Need or Requirement may be appropriate for more than one class of users. If so, we generally state the need or requirement for only that class of users considered to be the dominant source for the requirement (based on the outputs of the various workshops and written materials submitted during the Requirements Phase).

4 Each need or requirement is generally followed by a discussion. Examples are sometimes also given to illustrate the underlying problem that is the source of the need or requirement.

5 Not all user needs are reflected in stated Requirements because it is not clear that all such needs can be satisfied with the requisite minimal impact on implementations. Also, for some needs it is unclear whether the benefits of making a change are worth the potential costs. Such needs are reflected in *Study Topics*. The Mapping/Revision Team is expected to address Study Topics with less aggressive effort. Revisions addressing Study Topics must have a low negative impact.

1.4 General Guidelines for the Mapping/Revision Team

1 The following guidelines provide overall guidance to the Mapping/Revision Team. It is recognized that some of the guidelines are in tension with others; the Mapping/Revision Team must find satisfactory compromises.

2 G-1: ***Minimum Disruption:*** Changes shall minimize disruption to the Ada community while satisfying as many of the recognized needs as possible. In particular, this guideline covers the following criteria:

3 • **Implementation Ease:** Changes shall be relatively easy to implement with existing compiler technology, both individually and collectively.

4 • **Execution Efficiency:** Changes shall be efficiently implementable.

5 • **User Need:** The extent of a change shall not be out of proportion with the degree of user need for the change.

- **Upward Compatibility:** In general, any legal Ada 83 program must also be a legal Ada 9X program and must have the same effect allowed or required by Ada 83. However, to improve the portability of Ada programs, and to satisfy other requirements, it may be necessary to eliminate some effects allowed by Ada 83 or to make some Ada 83 programs illegal. In these cases, the following guidelines shall apply:

  - No implementation-dependent behavior allowed in Ada 83 shall be excluded in Ada 9X if that behavior is actually supported by a significant number of implementations and is important to a significant number of users.

  - No Ada 83 program shall be illegal in Ada 9X unless it is relatively easy to correct the illegal program or it is expected that very few existing legal programs will be affected (since the programs are either pathological or erroneous).

  - Since an Ada program should not, in general, rely on the raising of predefined exceptions (e.g., PROGRAM_ERROR), Ada 83 rules requiring the raising of a predefined exception in a given situation can be modified without undue concern for upward compatibility.

G-2: **Reliability/Safety/Maintainability:** Support for the reliability, safety, and long-term maintainability of large, complex systems shall take precedence over convenience of Ada coding. For example, this means greatly favoring statically checkable rules over rules checked at run-time, and favoring run-time checks over uncheckable rules (erroneous behavior).

2 General Requirements

There are a number of revision requirements that almost go without saying, since the revision of any language standard must at least address the small defects that have been discovered during the use of the standard. In addition, the overall structure and style of the revised standard present fundamental issues that must be addressed by the requirements. These basic requirements are addressed here. 1

2.1 Presentation Requirements

User Need 1 — Improve Wording to Reflect the Intended Meaning: A number of commentaries on the existing standard have been approved by the Ada Joint Program Office (AJPO) or the ISO Working Group responsible for the standardization of Ada (ISO-IEC/JTC1/SC22/WG9). The commentaries serve to correct unintended implications of the wording of the current standard and to point out places where some users have had difficulty in understanding the intent of the standard. To minimize confusion among Ada programmers and implementers, the wording of the standard should be revised, where necessary, to reflect these corrections. 1

Requirement 1.1 — Incorporate Approved Commentaries: Incorporate the approved Ada Commentaries into the revised standard to the extent that they do not conflict with changes dictated by other Requirements and to the extent that the commentaries indicate that it is appropriate to make the wording of the standard clearer. 2

Discussion: WG9's Ada Rapporteur Group (ARG) has been developing a report (the *Ada Commentary Integration Document* (ACID)) that reflects the impact of the approved commentaries by suggesting revised wording for appropriate sections of the standard. ACID would be a good starting point for satisfying this requirement. Of course, some of the approved commentaries may be inconsistent with changes suggested by the Mapping/Revision Team to meet other requirements, and in this case, the commentary should not be incorporated into the revised standard. In addition, since the process for developing the commentaries was limited to maintenance changes, some of the solutions proposed by the commentaries may not be appropriate given the greater latitude for change allowed by the Ada 9X revision process. In short, the approved commentaries should be reflected in the revised standard only when the intent of the commentary is consistent with the overall nature of the Ada 9X revisions. 3

Requirement 1.2 — Review Other Presentation Suggestions: Examine Ada Commentaries in the "Presentation" class for suggested wording and formatting improvements. Similarly, examine Revision Requests that point out areas where wording or other presentation aspects can perhaps be improved. Incorporate appropriate suggestions into the revision. 4

Discussion: Commentaries in the "presentation" class have not generally been formally approved by the Ada Rapporteur Group and WG 9, but some of them make useful suggestions for improving the presentation of the standard. Similarly, the following revision requests point out 5

areas where the clarity of the wording can perhaps be improved: RR-0167, RR-0204, RR-0206, RR-0260, RR-0267, RR-0274, RR-0281, RR-0292, RR-0298, RR-0301, RR-0305, RR-0309, RR-0350, RR-0436, RR-0500, RR-0501, RR-0502, RR-0757, RR-0769.

6 **User Need 2 — Maintain Continuity with the Existing Standard:** Since most users of Ada 83 are intimately familiar with the structure and format of the existing standard, it is important that the revised standard retain that structure and format.

7 **Requirement 2.1 — Maintain Format of Existing Standard:** The format of the Ada 9X standard shall be similar in style and structure to the existing standard. Chapter and section headings and numbers shall be retained to the largest extent possible. The presentation style shall also be retained.

8 **Discussion:** Requirement 1.1 and Requirement 1.2 direct the Mapping/Revision Team to address wording difficulties that have been discovered by Ada users over the years. In general, however, the structure and style of the standard should be retained by Ada 9X in order to preserve continuity with Ada 83. Maintaining the same document structure and style will help to minimize the impact of changes on the user.

9 **User Need 3 — Use of the Standard in User Documentation:** Compiler vendors and other organizations have found it helpful to provide annotated versions of the standard, e.g., to incorporate vendor implementation decisions or to reflect approved commentaries. Providing a machine-readable version of the standard facilitates such use.

10 **Requirement 3.1 — Machine-Readable Version of the Standard:** A machine-readable version of the revised standard shall be made available using a standard markup language such as SGML.

11 **Discussion:** Providing a version of the standard that contains formatting commands will make it easier to incorporate its content into data bases. In addition, some implementers and other individuals have produced annotated versions of the current standard that are helpful to users. This would have been easier to do if the standard had been made available with machine-readable markup codes.

2.2 Efficiency, Simplicity, and Consistency

1 **User Need 4 — Minimize Costs of Unused Capabilities:** In general, simple uses of existing or new language features should not incur unusual compilation or execution costs.

2 **Requirement 4.1 — Reduce Deterrents to Efficiency:** Examine combinations of language features that seem to incur unusually large compilation or execution costs in relation to benefits to users. Reduce these costs where possible. This requirement applies both to features in the existing standard and to proposed changes for Ada 9X.

User Need 5 — Ease of Learning: Language rules should be easy to learn and use correctly.

Requirement 5.1 — Understandability: Examine those rules that have proven to be confusing or error-prone to users and attempt to improve them. Appendix Section A.1 on page 46 includes some of the rules that should be considered for revision.

User Need 6 — Generality: Sensible uses and combinations of language constructs should be allowed.

Requirement 6.1 — Minimize Special-Case Restrictions: To the extent consistent with Requirement 5.1, rules barring special uses and combinations of language constructs shall be eliminated. Appendix Section A.2 on page 47 lists some of the rules that should be considered for revision.

Discussion: Experience with Ada 83 has revealed a number of opportunities for simplifying, generalizing, or otherwise revising the language rules so as to avoid unusually large compilation or execution costs and to make the language easier to learn. The above Needs and Requirements direct the Mapping/Revision Team to reexamine the rationale for the original language rules in such areas. If the reasons are no longer applicable, appropriate corrections should be made, consistent of course with the general guidelines G-1 and G-2.

It should be noted that the above requirements directly conflict with each other. Minimizing costs of unused capabilities is typically satisfied by having special case restrictions, thus violating User Need 5 (Ease of Learning) and User Need 6 (Generality). User Need 5 can be satisfied by making the language very general and orthogonal, risking inefficiencies that would violate User Need 4, or by simplifying and restricting the language, which could violate User Need 6. The Mapping/Revision Team must find an appropriate balance among these conflicting requirements.

2.3 Error Detection

User Need 7 — Minimize Consequences of Programmer Error: Violations of language rules should be detected by pre-execution checks whenever practical. When such checks are not practical, violations should be detected by run-time checks, so exception handlers can recover from the error, or so programs can be abandoned in a safe and predictable manner. Whenever practical, violations should not place a program in an undefined state with unpredictable consequences.

Study Topic 7.1 — Improve Early Detection of Run-Time Errors: If an implementation is able to detect that a run-time program error is present (i.e., that a compilation is erroneous or that a predefined exception will necessarily be raised), the implementation should be allowed to reject the compilation (i.e., not add it to the program library).

3 **Discussion:** Implementations can sometimes detect at compile-time that the execution of a program will be erroneous or will raise a predefined exception such as CONSTRAINT_ERROR or STORAGE_ERROR. Such situations, especially those that can be easily detected at compile-time, are almost always programmer errors. In general, programmers want as much help in detecting possible problems as they can get. Moreover, they want to be able to count on the nature of such help as they port applications among compilers. Ada 9X can help achieve these goals by specifying more clearly that implementations are allowed to reject compilations containing probable program errors of the kind specified in the requirement.

4 Of course, implementations today are allowed to issue "warning" messages in these situations, and they can provide an optional mode of operation in which certain certain kinds of warnings cause compilations to be rejected, but this potentially useful behavior is not uniformly provided. At the least, Ada 9X should make clear that program rejection is allowed under such circumstances. For some applications (such as safety critical and trusted system applications), a special annex might specify that implementations definitely detect certain kinds of these situations, requiring that a compilation be rejected if such situations are detected.

5 On the other hand, there are some compilations that should not be rejected even though they appear to contain these kinds of errors. For example, a program might appear to reference an uninitialized variable but the programmer may know that the variable is given a valid value as a result of some external action, e.g., this could be the case for memory-mapped I/O. In such cases, there must be some way for programmers to ensure that the compilation is not rejected.

6 This requirement is classified as a "Study Topic" because it is not completely clear how far the Ada 9X standard should or can go in specifying situations that *must* be detected. Such a list might be application-dependent, e.g., trusted and safety-critical systems might want to impose more severe constraints on error detection than would be considered cost-effective for other application areas. Moreover, not all such errors can be easily detected at compile-time. Requiring that all compilers detect certain errors of this kind could impose an unacceptable implementation burden on some compilers, e.g., inexpensive compilers that are intended for the education market.

7 **Requirement 7.2 — Limit Consequences of Run-Time Errors:** Run-time violations of language rules shall have narrowly defined consequences. Whenever practical, a violation shall result in the raising of an exception. When the raising of an exception is not practical, the possible consequences of the violation shall be few, confined to limited aspects of the computation state, and defined in the Standard. This definition shall be in terms of those aspects of the computation state addressed elsewhere in the Standard. To the extent possible, catastrophic consequences shall be excluded.

8 **Discussion:** In Ada 83 most rules are enforced either by compile-time or run-time checks. For a small number of rules, neither compile-time nor run-time checks are practical. Violation of such rules has unpredictable consequences. Execution is said to become "erroneous" when such a rule is violated.

The consequences of erroneous execution can be devastating. Arbitrary storage locations (possibly including hidden run-time data structures) may be overwritten, or control may be transferred to a location that does not correspond to any declaration or statement in the source program. While such consequences are allowed by Ada 83 rules, they are easily avoided in many cases. For example, execution is rendered erroneous by a subprogram call whose effect depends on the parameter-passing mechanism, but for any reasonable implementation only a small number of precisely defined effects are possible — the call proceeds normally with each parameter passed either by copy or by reference. In other cases, the consequences of erroneous execution could be made more predictable at modest cost. 9

2.4 Control Implementation-Dependent Choices

User Need 8 — Uniformity of Compiler Behavior: Because Ada is intended to be a language that enhances the portability of software, Ada programs should behave in the same way with different implementations. In some cases, Ada 83 has deliberately permitted implementation freedom to reflect expected variations in target and operating systems environments. For example, the range of the predefined type INTEGER is not specified, since the intention is that this reflect the appropriate target type. There is a need to minimize these differences, since in practice they cause difficulty in porting programs. 1

Requirement 8.1 — Minimize Unnecessary Implementation Dependencies: The Mapping/Revision Team shall identify allowed implementation dependencies and ensure that those remaining are appropriately justified by considerations of target variations. 2

Discussion: In some cases, Ada 83 permits variations in anticipation of operating system and target dependences where experience has shown that uniformity could be achieved. The Uniformity Rapporteur Group (URG) has been systematically investigating such variations and is in the process of producing uniformity recommendations. For example, Ada 83 specifies that the status of unclosed files at program termination is undefined, but the URG recommends that files be required to be closed on program termination. In some cases, it is not possible to completely eliminate variations among implementations, but their effect can be minimized. For example, another URG recommendation suggests that the predefined INTEGER type have certain minimum characteristics. Another area where uniformity can be improved is with respect to Ada implementations on similar or identical targets, where unnecessary variations should be minimized. For example, the format of TEXT_IO files in Unix implementations should be standardized (perhaps in accordance with the POSIX Ada binding that is under development). 3

3 Requirements for International Users

Ada is an international language. However Ada 83 does not adequately address the need for handling non-English character sets. In addition, the Ada 83 syntax is inconvenient for use by non-English speaking programmers. These issues were identified during the ISO standardization process for Ada 83 and a commitment was made by the ISO group responsible for the Ada standard that the character set issue would be addressed when the ISO standard was next revised. Since a goal of the Ada 9X effort is to maintain Ada as an international standard, this ISO committment must be addressed by the Ada 9X effort.

3.1 International Character Sets

The Ada 83 character set represents a rather small subset of the graphic symbols used in natural languages throughout the world. This fact complicates writing programs and handling character and string data in non-English-speaking communities. Enhancement of Ada's character set will extend the use of Ada and facilitate software development world wide.

User Need 9 — International Character Set: It is necessary to adjust the character set of Ada 9X to make it useful for international use. There are three specific needs:

1. It must be convenient in Ada 9X to write applications that are able to deal with character sets other than English.
2. All Ada programs must be accepted by all Ada compilers — that is, there must be no national variants.
3. It must be convenient for programmers whose native language is not English to write Ada programs.

Discussion: It follows from item 1 that the alphabet of character and string literals must include additional characters, and that there must be functionality equivalent to the present TEXT_IO for such alphabets.

It follows from item 2 that any Ada lexical element accepted by one validated Ada compiler must be accepted by any other.

A minimum response to item 3, already reflected in approved AI-00339, is that non-English characters must be permitted in comments; it is desirable that they also be permitted in identifiers. However, various meetings with members of the international community have indicated that the translation of reserved words is not considered to be a requirement, and indeed such a translation would be difficult to achieve in a manner that did not conflict with item 2.

Requirement 9.1 — Base Character Set: The Ada 9X type STANDARD.CHARACTER shall be based on the ISO 8859 [9] (8-bit) standard for character representation in the same way that the Ada 83 character set is based on the ISO 646 [8] (7-bit) standard.

10 **Discussion:** This requirement introduces the potential for incompatibility with existing programs (for example, in constructs such as case statements and array aggregates that might require choices for every value of type CHARACTER), but it has been widely recognized that an 8-bit standard is more appropriate as the base character set. ISO 8859 actually defines a number of different 8-bit character sets, and the Ada 9X design must take this fact into account. For most purposes, the external graphic representation of the character set does not affect the implementation, but there are issues such as the collating sequence which must be considered since the proper "lexicographic" ordering required by Ada 83 for the type STRING cannot be achieved simply by using the coded representation of the characters comprising a string.

11 The next requirement addresses the issue of larger character sets, in particular those of Asian countries including Japan, China, and Korea that use very large character sets. Ada 9X must provide additional character types to support these and other international languages (see Brender's report on character set issues for an analysis of the issues posed by additional character types [6]). Where possible, Ada 9X should be consistent with developing ISO standards in this area.

12 **Requirement 9.2 — Extended Graphics Literals:** Ada 9X shall permit the use of non-English characters, including those from international character sets with more than 256 graphic symbols, in character and string literals.

13 **Requirement 9.3 — Extended Character Set Support:** Ada 9X shall provide input/output facilities and other operations for extended character sets (specifically, for ISO 8859 and for international character sets with more than 256 graphic symbols) which are comparable to those provided for the base character set.

14 **Requirement 9.4 — Extended Comment Syntax:** The use of international character sets (including those with more than 256 characters) shall be permitted in comments in Ada 9X source code.

15 **Discussion:** Although the inclusion of extended characters in comments raises no conceptual problems, there is an issue of source representation which needs study. Ada 83 does not specify the physical form of source programs, and this approach could be continued in Ada 9X. However, the issues of source representation must be further studied. Secondary standards in this area may be desirable, since the mechanisms for representing text that uses large character sets are not well standardized in practice.

16 **Study Topic 9.5 — Extended Identifier Syntax:** To the extent practical, Ada 9X shall permit the use of extended character sets in identifiers.

17 **Discussion:** The identifier issue is somewhat complex. Among issues to be considered is the applicability of the upper-lower case equivalence rule. The various parts of the 8859 standard have inconsistent requirements in this area. A general solution providing completely satisfactory use of foreign identifiers in all languages is likely to be unacceptably complex; some appropriate

compromise is needed in this area. One suggestion is that at least Latin-1 be permitted, which fully meets the needs of all Western European countries. It is reasonable to move at least this far — the question of how much further to go needs study.

It is highly desirable that the Ada 9X solutions in this area be compatible with the recommendations of the ISO WG9 CRG (Character Rapporteur Group), since the CRG's recommendations are likely to be used as guidelines for accomodating international character sets in existing Ada 83 implementations. 18

4 Requirements for Support of Programming Paradigms

4.1 Subprogram Variables

User Need 10 — Subprogram Variables: There is need to manipulate subprograms as run-time values. 1

Study Topic 10.1 — Subprogram as Objects: Ada 9X should provide subprogram-valued variables and permit general use of such objects, doing so in a way that provides safety while incurring no efficiency cost on code that does not use this feature. 2

Discussion: Meeting the goal of this Study Topic implies that Ada 9X support (1) the declaration of types whose values are subprograms, (2) creation and manipulation of variables of such types, (3) passing values of such types as subprogram and entry parameters, (4) returning values of such types as the value of a function, and (5) application of a value of such a type to a conforming argument list. 3

This Study Topic is unusual in this document in that it is presented as a solution rather than in terms of underlying needs. Although most such needs can be satisfied by simpler changes to the language, it is nonetheless the opinion of the Requirements Team that this solution is the best available for these and other needs that have been identified during the revision process. 4

- Some applications (such as X-windows) use a call-in/call-out mechanism to pass a subprogram object to another subprogram, the former to be invoked at some time in the future. Although it is possible to meet this need in Ada 83, it cannot be done in a portable manner. 5
- Scientific applications need to supply a numeric function, for example to an integration routine. Use of generic instantiation is inadequate in those applications in which the function is not known until execution time. 6
- This feature provides a relatively simple way to get some of the functionality of object oriented programming. 7
- In some applications major mode changes are required, sometimes with minimal execution overhead. For example, an aircraft experiencing an emergency might want to cease expending computer resources to control an air conditioning unit. 8
- Use of subprogram variables has been a useful and well-proven paradigm for 20 years or more; its use is known to simplify the solution to many programming problems. 9
- It is often desirable to determine at execution time the identity of the subprogram that will be called to fulfill a given role. 10

4.2 Storage Management

1 In many applications the programmer requires precise control over the allocation and deallocation of data storage. In the extreme (and realistic) case of a long-running application, the ongoing allocation of storage that is not reclaimed is unacceptable, whether the storage is allocated explicitly by the programmer or implicitly by the run-time system. For example, if the compiled code uses the heap to hold the value of a function (perhaps of an unconstrained type), then the compiler must either recover that storage (desirable) or provide the programmer a way to do so (acceptable).

2 *Required* is the ability to program in a style that precludes storage leakage; *desired* is the ability to do so while preserving abstraction. If a package exports an abstract data type, it is desirable (but not required) that the application be able to reclaim associated storage from within the abstraction.

3 **User Need 11 — Control of Storage Use:** Ada 9X must permit a style of programming in which no storage is used in a manner that makes it impossible for the programmer to reclaim it.

4 **Requirement 11.1 — No Loss of Storage:** Ada 9X must provide mechanisms so that an application is able to reclaim *all* heap storage that is allocated. This requirement implies that implementors must document all ways in which the implementation uses the heap, including space used for tasks, and provide for each a way to reclaim the storage.

5 **Study Topic 11.2 — Preservation of Abstraction:** To the extent possible, Ada 9X shall meet Requirement 11.1 in a manner that permits preservation of abstractions.

6 Discussion: It is desirable for a programmer implementing an abstract data type in terms of dynamic data structures to be able to determine when memory used for that abstraction may be reclaimed. At a minimum, the programmer should be able to gain control whenever a value is copied and whenever a scope is deactivated. (Note that finalization and user-defined assignment appear to be needed here.)

7 Meeting this requirement provides an enhancement to assist the programmer in controlling storage deallocation. In Ada 83, it can be difficult to insulate a user of a type from the decision to use dynamic data structures for its implementation — unless garbage collection is provided — because there is no way for a type implementer to determine when an abstract object is being "unreferenced" so that appropriate steps can be taken to reclaim the memory that the object occupies.

8 Note that garbage collection may not be an acceptable solution in time-critical applications.

4.3 Construction of Large Programs

One of its principal design goals of Ada 83 was to ease the job of constructing large programs. In particular, these features contribute toward meeting this need:

- syntactic separation of unit specifications from unit bodies,
- a program library as a database of interface information about separately compiled units, and
- ensured safety of system builds in that all language rules are enforced across separate compilation boundaries and program binding is permitted only when none of the constituent units is obsolete.

However, an effect of these features in Ada 83 has sometimes been unacceptably large recompilation costs.

User Need 12 — Minimization of Compilation Costs: Users require that compilation costs during program development be minimized.

Requirement 12.1 — Minimize Recompilation Costs: The separate compilation facility shall be modified so as to minimize the need for a cascade of recompilations of dependent units after modifying a unit.

Where relevant, the separate compilation rules in Ada 9X should enhance the current Ada 83 support for safety of program binding and elimination of unnecessary recompilation.

Discussion: Experience with Ada 83 has uncovered a small number of areas that merit consideration for possible improvement in Ada 9X. One example is the implicit obsolescence of an optional package body when the package specification is recompiled. Under the current rules, a program bind is permitted, with the obsolete body excluded. If this is not the programmer's intent, the resulting program will not work correctly.

Potential enhancements could also be useful in minimizing recompilation. Recompilation of generic bodies should not require recompilation of the units containing instantiations. (Equivalently, it should be possible to compile a unit containing an instantiation before the generic body has been compiled.) It would also be useful if certain modifications to a unit (such as the addition of comments) did not invalidate dependent units.

Another issue that comes up in practice is porting large Ada systems across different Ada compilers. Implementations vary with regard to how they deal with separate compilation, for example whether and how multiple libraries are supported, and these differences have to be taken into account by Ada users. The Ada language requirements on library management intentionally provide wide freedom to the implementation, given the variety of host environments and the fact that the issues in this area are less understood than in other facets of the language. The tradeoff, however, is with ease of portability, and thus the Mapping/Revision Team should investigate this issue to see if it is useful to have more specific required library support.

12 **Study Topic 12.2 — Enhanced Library Support:** In the interest of promoting uniformity across implementations, the Mapping/Revision Team shall investigate which additional functionality, if any, should be required for library management. In particular, the issue of multiple libraries (for example, sharing a common specification across several libraries) should be addressed.

4.4 Object-Oriented Programming

1 **User Need 13 — Support the Object Oriented Programming Paradigm:** The Object Oriented Programming paradigm has been shown in recent years to have important benefits in software construction, particularly with respect to reuse and education. Ada 9X should provide support for that paradigm to the extent that it can do so without conflict with the other requirements of the revision.

2 **Study Topic 13.1 — Inheritance:** Ada 9X shall provide a mechanism to define *categories* to characterize a set of values with common fields and operations. It shall be possible to create objects of a given category just as it is possible to create objects of a given type. Ada 9X shall provide an inheritance mechanism whereby new categories can be defined based on existing ones, with additional fields and operations, or with different implementations of existing operations. It shall be possible to use an object or value a new (child) category anywhere objects or values of the original (parent) category could be used.

3 **Study Topic 13.2 — Polymorphism:** Ada 9X shall allow a form of reference to categories where the categories may be of any category type obtained from an original category by zero or more applications of the inheritance mechanism.

4 **Discussion:** The notion of a *category* described above corresponds to the notion of a *class* in object-oriented languages. We avoid the term *class* because it is used in a different sense in the Ada reference manual.

5 An important concept in software reuse is that a software component may be parameterized by a type. A parameterized component should work not only for a given type but also for any of a number of related types constructed from the original type. The advantage of obtaining types by construction is that there are broad conditions under which operations defined for the original type continue to be viable for constructed types. In the case of inheritance, this means that common underlying capabilities can be implemented and shared easily among multiple types in a layered fashion.

6 Because usual solutions require heavy heap usage, Study Topic 11.2 on page 18 becomes particularly relevant.

4.5 Generics

There are two broad areas of user needs. One is usability, including expressive power with general parameterizability as well as early detection of errors (at instantiation time, even if the body has not yet been encountered); the other is code space efficiency (code sharing). 1

These needs conflict, and Ada 83 offers one particular compromise. In the light of user experience, the Mapping/Revision Team should examine the tradeoffs that were made, identify which if any language improvements are required, and make the necessary revisions. Because of the conflicts between these needs, as well as the wide range of tradeoffs in response to these needs (including making no language changes at all), there are no formal requirements associated with the identified user needs. 2

User Need 14 — Support for Reusability: The design of a generic unit involves abstraction from particular instances to a general template, where the variability across the instances is reflected in the values passed as parameters at instantiations. A fundamental need, then, is for the language to supply sufficiently general generic parameterization to allow the definition of wide classes of reusable components, such as data structures (lists, queues) and mathematical and graphics packages. 3

Discussion: The so-called "contract model" (see the next paragraph) coupled with requirements in Ada 83 for staticness in a number of contexts have resulted in situations where generics do not provide all the expressive power that would be desired. For example, it is not possible to use an attribute of a formal generic parameter where a static value is required. Relaxing the contract model or the requirements for staticity, or enhancing the kinds of parameterization, would make it easier to write certain useful kinds of generics (for example, a generic math package where the precision of a local type depends on the precision of formal type). 4

User Need 15 — Independent Use and Implementation of Generics: A generic specification ought to serve as a "contract" between the users and implementers of a generic unit. An appropriate contract promotes *separation of concerns*, allowing a user to instantiate a generic unit without regard to the contents of the generic body and allowing an implementer to write a body without regard to the ways in which the unit will be instantiated. Ideally, once a contract is established and the generic specification is entered into the program library, it is possible for users and implementers of the generic unit to proceed independently and in parallel, compiling their products in any order. The language supports the contract model to the extent that it is possible to ensure the compatibility of a generic instantiation with the corresponding generic body by individually ensuring the compatibility of each with the generic specification. 5

Discussion: In a small number of situations, it is possible in Ada 83 to have the legality of a generic instantiation depend on how the generic formal parameter is used. One example is passing an unconstrained type as an actual parameter to a formal private or limited private type. Such an instantiation is correct only if the generic template contains no declaration of a variable having 6

that type, since it is illegal to declare a variable having an unconstrained type. Tightening the contract model would eliminate such anomalies.

- Note that the discussion following Requirement 12.1 on page 19 calls for tightening the contract model so as to minimize recompilation costs.

- **User Need 16 — Code Space Compactness:** A direct implementation of generic instantiation results in replicating the code for the expanded template. In some environments such code replication is undesirable, and it is preferable to have a single version of the template, with run-time parameterization to identify instantiation-specific information.

- **Discussion:** Two principal situations are candidates for code sharing. One is a generic template parameterized by subprogram (for example, a generic procedure for mathematical integration). This is the Ada 83 style for accomplishing what in other languages is done by passing subprograms as run-time parameters; such an implementation is expected by many users. The second situation is a generic template taking, for example, a formal access-type parameter. Instantiations with types sharing the same machine representation can often share the same code. Such an optimization is useful for space efficiency.

5 Real-Time Requirements

This chapter deals with requirements posed by real-time applications, i.e., applications in which it is important to have precise control over *when* some action is taken. Other requirements of potential interest to the real-time community are addressed in Chapter 6, Requirements for System Programming and in Chapter 7, Requirements for Parallel and Distributed Processing. 1

5.1 Time

User Need 17 — Time Measurement: Real-time systems typically need to measure time in two distinct ways [5, p. 124]. Computations of time-based physical parameters such as velocity and acceleration depend on the measurement of *elapsed time*. The key property of elapsed time is that its value increases monotonically at a known rate (from some convenient starting point, e.g., mission start time). 1

On the other hand, for interfacing with external entities such as other processors, devices, and human operators, the *time of day* is needed. The time of day does not necessarily increase monotonically at a fixed rate, since it may have to be adjusted to account for time zones or leap seconds, or to synchronize with other clocks on a local area network. Such adjustments may cause the time-of-day value to retrogress, while retrogression is never allowed for measurements of elapsed time. 2

The precision of the two kinds of timing measurement may differ; a clock used to control and monitor elapsed time may be updated at a faster rate than the time-of-day clock when elapsed time measurements are used to control physical processes in tight feedback loops. 3

Requirement 17.1 — Support Elapsed Time Measurement: Ada 9X constructs that deal with time shall support two abstractions of time — monotonically increasing elapsed time and the time of day. The time of day abstraction shall be compatible with the Ada 83 package CALENDAR and may be maintained with different precision from that of elapsed time. 4

Discussion: The time defined in Ada's package CALENDAR is the time of day (see AI-00195). While real-time applications must sometimes be synchronized with the time of day, more frequently such applications need to use measurements of elapsed time. Baker [4, 5] discusses in more detail why both kinds of time measurement are needed. 5

User Need 18 — Periodic Tasks: Executing tasks at precise periodic intervals is an important requirement for a significant number of real-time systems. 6

Requirement 18.1 — Precise Periodic Execution: Ada 9X shall provide a standard way to control the periodic execution of a task with respect to elapsed time or the time of day. The method provided shall allow a task's period to be changed during the execution of a program and shall ensure that the task becomes schedulable for execution at precisely the specified time. 7

8 **Discussion:** The current **delay** statement is not generally suitable for precise periodic task scheduling because the computation of the necessary delay can be preempted at critical moments, leading to inaccurate periodic task execution (see [4]).

9 This requirement is phrased neutrally with respect to whether precise periodic task scheduling is provided by a pragma, a special call to the run-time system, or new syntax. In any case, performance constraints on implementations targeted to real-time applications will have to be specified to ensure that the specified solution is supported in usable fashion. Indirect solutions to this requirement could also be satisfactory. For example, the Third International Workshop on Real-Time Ada Issues [5] suggested that a **delay until** statement could be used to achieve this requirement. Another approach is being explored by the Ada Run-Time Environment Working Group (ARTEWG) with their Catalogue of Interface Features and Options (CIFO), which is currently under development.

10 **User Need 19 — Overrun Detection and Response:** Real-time programmers need ways of specifying deadlines and ways of specifying what action is to be taken when a task overruns its deadline. Among the actions typically taken are: log the overrun and continue processing, stop the overrunning task from continuing its execution (restarting it at an appropriate point and time), temporarily changing the task's rate of execution, etc.

11 **Requirement 19.1 — Ensure Detection of Missed Deadlines:** Ada 9X shall allow programs to detect that a task has missed a deadline and to initiate a corresponding action.

12 **Discussion:** In some applications, it is important that an overrun be detected quickly after it occurs so processing can be stopped and other processing started. This requires an asynchronous transfer of control (see Requirement 21.1 on page 26).

5.2 Task Scheduling

1 **User Need 20 — User-Controlled Scheduling:** Scheduling algorithms determine when a task is selected for execution. Scheduling decisions are associated with a variety of different Ada constructs — entry queues (selection of the next call to be serviced), selective waits (selection from among calls waiting at open alternatives), attempts at task synchronization via entry calls (how is the priority of the called task affected), priority and/or deadline of a given task in relation to other tasks, etc. Specialized scheduling algorithms giving improved performance are sometimes needed to take advantage of an application's expected use of tasking constructs. Since the same scheduling algorithm is not suitable for every real-time application, application developers must be able to determine the algorithm to be used for a particular application.

2 **Requirement 20.1 — Support Alternative Scheduling Algorithms:** Ada 9X shall allow real-time programmers to determine the scheduling policies and algorithm(s) to be used for a particular program.

Discussion: This requirement intentionally avoids specifying how scheduling policies will be specified by the user. One possibility is that Ada 9X will specify a set of standard scheduling policies plus a means of selecting which policy/algorithm is to be used for a given program. Another approach is for Ada 9X to specify low-level primitives or a standard run-time interface from which scheduling policies and algorithms can be built. 3

Many in the real-time community favor the low-level or standard interface approach because of the flexibility it can provide. But it is not yet clear that a standard interface or set of primitives can be exploited efficiently for all potential target machines — any given interface always seems to be biased in favor of some class of hardware architectures or some anticipated pattern of use which may not be right for a given application. Even compiler vendors sometimes find it difficult to stick to a self-imposed standard run-time interface when highly efficient performance or specialized optimizations are demanded. 4

The advantage of having Ada 9X specify some standard scheduling policies (e.g., in a real-time annex of the standard) is that the policies could provide a standard framework for designing and implementing real-time applications. The potential disadvantage, of course, is that applications requiring some other policy or some adjustment to a scheduling algorithm may be unable to use Ada tasks. Moreover, there is little agreement on what such policies must be. Nonetheless, specifying some standard scheduling policies could help improve the portability of real-time applications. 5

Given that Ada's current scheduling rules are relaxed to clearly allow the use of alternative scheduling algorithms, some feel that sufficient flexibility is provided simply by giving application developers access to vendor-supplied Ada run-time systems. This is not an entirely satisfactory solution since a given compiler vendor may not provide compilers for all the target configurations of interest to the application developer. In such a case, an application developer prefers to develop a portable executive that can be interfaced efficiently with different Ada compilers. 6

Requirement 20.2 — Efficient Support for Common Tasking Idioms: Ada 9X shall ensure that common tasking idioms are supported efficiently. 7

Discussion: There are a number of common uses of Ada tasking constructs that must be supported with particular efficiency and predictability for real-time applications. For example, the abort statement may be unusable in real-time applications if it does not take effect until the aborted task reaches a synchronization point. The utility of the delay statement depends on the accuracy of its implementation. In high performance applications, interrupt entries need to be implemented without any scheduler intervention at all. It may not be possible to use tasks to ensure synchronized access to data structures unless the entries for storing and retrieving data are implemented with special efficiency. 8

- Application developers also sometimes restrict their uses of tasking constructs in ways that permit special execution efficiency. For example, tasks in embedded real-time systems are often declared only at the library package level, meaning there is a fixed number of tasks and there are no dependent tasks. Under these conditions, the performance of various scheduling operations can be significantly improved. For example, task abortion and termination is simplified because there are no dependent tasks.

10 Some tasking idioms are discussed in the Proceedings of the Third International Workshop on Real-Time Ada Issues [10]. For example, a monitor task (also called a passive task) is a specialized tasking idiom that allows rendezvous to be performed for essentially the same cost as a procedure call. To achieve this performance, different implementations today impose different constraints on the structure and content of such tasks (e.g., the task body is required to consist of a single select statement in a single loop, and it can only call other passive tasks). Ada 9X could improve this situation by specifying standard restrictions that ensure no scheduler intervention is required to execute a rendezvous (assuming the entry call is not blocked) and by requiring that implementations document tasking performance parameters.

5.3 Asynchronous Transfer of Control

1 **User Need 21 — Asynchronous Transfer of Control:** There are many situations where it is useful to terminate one sort of processing and continue processing in a new place, based on the asynchronous occurrence of some event. For example, it is sometimes necessary to stop a process in response to a user interrupt because there is no longer any need for the activity and resources must be released to do something else (e.g., aborting a bombing run to engage in a defensive maneuver). Similarly, it is sometimes necessary to stop a process when it has taken too long so a short-cut algorithm can be applied instead.[1]

2 **Requirement 21.1 — Asynchronous Transfer of Control:** Ada 9X shall allow the execution of a sequence of statements to be abandoned in order to execute a different sequence within the same task. The event causing this transfer can be an external interrupt (for example, a timer) or a call from another task. It shall be possible to abandon the execution quickly, even if execution is suspended at an entry call or an accept statement. In balancing the need for immediate response with the need for safe and predictable execution (e.g., the ability to release resources held by the abandoned code) (see Guideline G-2), it must be possible to identify certain sections of code in which a request for asynchronous transfer of control will not be honored.

3 **Discussion:** Events requiring an asynchronous transfer of control can arise under conditions where they must be handled quickly, within strict time constraints. It must be possible to stop the computations of certain tasks quickly. Suspension is not enough because when a task resumes execution it must continue at a different execution statement. (For example, if an autopiloted

[1] The text of this User Need is drawn from RR-0083, RR-0106, and RR-0384.

landing is aborted, the responsible tasks should be restarted from the beginning, with fresh data, the next time a landing is attempted.) In some cases, the transfer of control must be immediate.

The design of Ada in 1980 proposed the use of an asynchronous exception to effect an asynchronous transfer of control. This mechanism was removed from the final design because various significant difficulties were discovered (see RR-0196 and [11] for a full discussion). The relevant difficulties have been mentioned in the above requirement to preclude re-introduction of this mechanism. 4

Requirement 21.2 — Asynchronous Transfer of Control Performance Standards: Ada 9X shall specify performance parameters and constraints affecting the implementation-dependent timing properties of the Ada 9X asynchronous transfer of control mechanism. In particular, real-time implementations shall be required to implement the transfer of control "immediately" for some reasonable definition of this term. (The wording of this definition should not allow the target sequence of statements to run indefinitely, such as until it reaches a synchronization point.) 5

Discussion: It is expected that performance parameters and constraints affecting the usability of constructs intended primarily for use in real-time applications will be specified in a special Ada 9X language annex aimed at helping to ensure the usability of real-time implementations. 6

5.4 Asynchronous Communication

User Need 22 — Asynchronous Message Passing: The ability to send data to a task without waiting for it to be received is an important and common paradigm in real-time programming. For example, device drivers in an embedded system must be able to send data to other tasks without any scheduler intervention that might cause interrupts to be missed. Similarly, non-blocking calls are used when sending large amounts of information to relatively slow recording devices. 1

The message passing paradigm is also appropriate for computer systems that do not have shared memory and for massively parallel systems. Finally, in some applications, messages with different priorities need to be sent to the same task; the messages need to be processed in priority order. 2

Real-time programs could be easier to understand and maintain if the non-blocking call paradigm were directly expressible in the language. Most real-time operating systems provide direct support for non-blocking communications, and real-time programmers are accustomed to having this capability available. Non-standard, implementation-dependent mechanisms are currently being used to support asynchronous communication needs. To reduce implementation dependencies and increase maintainability and portability, Ada 9X needs to specify standard asynchronous communication mechanisms. 3

Requirement 22.1 — Non-Blocking Communication: Ada 9X shall support non-blocking, asynchronous communication between tasks in an Ada program. In particular, from the view- 4

point of the sending task, the maximum time to execute an asynchronous call shall be independent of the state of the receiving task and shall be essentially the same as the time required to read and store the data.[2]

- Discussion: Agent tasks that accept messages to be queued are the standard Ada construct for attempting to support non-blocking communications. Since all agent tasks have the same priority, service order is not guaranteed by Ada 83 rules. If agent tasks are to be used for this purpose, the overhead that is apparently involved in creating and terminating them must be optimized out of existence since fast service is especially important when an interrupt handling routine needs to store data very quickly and predictably in a buffer.

- From a programmer viewpoint, providing the capability with a standard package would probably be a satisfactory solution, but Ada's current generic unit capabilities make the use of such a package awkward (see [3]). It is possible that other Ada 9X changes to generics (see Section 4.5) would make it easier to provide an efficient message passing package.

- User Need 23 — Asynchronous Broadcast: The ability to broadcast messages to a group of tasks (without waiting for a reply) is sometimes necessary. Similarly, the ability of a receiver to determine its own eligibility to receive such a message is sometime useful.

- Study Topic 23.1 — Asynchronous Broadcast: The Mapping/Revision Team shall study the possibility of supporting asynchronous communication from a single sender to multiple receivers as well as the ability of potential receivers to determine their eligibility to receive a message.

[2]Assuming there is space to store the data being sent.

6 Requirements for System Programming

6.1 Machine Operations

User Need 24 — Machine Operations: Essentially all modern CPUs provide a wide complement of unsigned arithmetic, logical, and shifting operations on register-length data, and many coding algorithms (for example, checksums and cyclic redundancy checks) are formulated in terms of these operations. An efficient abstraction is needed that provides efficient access to these operations in such a way as to facilitate realization of them by the underlying basic ALU capabilities of modern CPUs. 1

Requirement 24.1 — Unsigned Integer Operations: Ada 9X shall provide an abstraction that will map efficiently onto the wide complement of unsigned arithmetic, logical, and shifting operations on register-length data provided in most modern architectures. 2

6.2 Data Interoperability

User Need 25 — Data Interoperability: Ada 83 does not specify the exact representation of data, either in memory or on external data files. The user needs to be able to specify and control data representation for the purpose of exchanging data with other languages via pragma INTERFACE, and also for reading and writing external data files to be processed by other programs, which might be written in Ada (perhaps compiled by a different implementation) or in some other language on the same target machine or even on some other target. 1

Study Topic 25.1 — *Data Interoperability*: *Ada 9X shall provide the means to specify the exact* representation of data in memory and on external files so that interoperability with external components can be guaranteed. 2

Discussion: Many applications require the ability to interface via memory with system components external to an Ada program. Often the format of this memory is not under the control of the programmer. Further, the system-level design does not always take the Ada object model into account. For example, the "type" of a block may vary dynamically with no explicit discriminant indicating the current "view". In these situations, the Ada 83 representation clauses alone are inadequate to specify fully the representation. Representation specifications in Ada 83 do not provide a sufficiently fine level of control. For example, there is no control over the bit/byte ordering — such control is required in order to deal with the conflicting representations between "little-endian" and "big-endian" representations. 3

6.3 Interrupt Entry Binding

1 **User Need 26 — Interrupt Handling:** Programmers of embedded applications need a way to associate interrupt handling code with hardware interrupt structures, a way to deal with the specific interrupt mechanics of the hardware, and a way to quickly notify tasks awaiting the event signified by the interrupt.

2 **Requirement 26.1 — Interrupt Binding:** Ada 9X shall provide a means to associate (and reassociate) interrupt handling code with interrupts at run time. It shall be possible to associate different instances of an interrupt handler with different interrupts.

3 **Discussion:** Ada 83 provides a means to bind interrupt entries to interrupt locations when a task declaration is elaborated. This mechanism, however, does not allow an interrupt binding to be changed during execution. An interrupt binding may need to be changed dynamically for a variety of reasons, e.g., to recover from a hardware fault or to respond to an operational mode change.

4 **Requirement 26.2 — Interrupt Mechanics:** Ada 9X shall provide mechanisms to handle mechanics of interrupt hardware in an abstraction that is compatible with the Ada execution/tasking model. At a minimum, these mechanisms shall:

5 1. Permit the interrupt handling code to mask its triggering interrupt, e.g., while it is manipulating hardware which may cause the interrupt to occur or while it is not ready to accept or handle the interrupt.

6 2. Permit handling of interrupts without using the resources of the software scheduler, unless the interrupt handler interacts with Ada tasks.

7 3. Permit the interrupt handling code to handle nested interrupts.

8 **Discussion:** An interrupt handler is the mechanism through which an Ada application quickly and efficiently transforms a physical event, such as an IO complete or an illegal instruction, into a logical event. Ada's method for handling interrupts needs to be improved in various ways:

9 • It would be convenient to be able to use arrays of tasks to manage hardware interrupts, but since interrupt bindings can only be specified in task declaration blocks, it is virtually impossible to create such arrays.

10 • Tasks may lose interrupts because the task entry/rendezvous mechanism is often not supported by hardware systems. If the interrupt occurs before the task arrives at the accept, the interrupt is not required to be queued, and in fact many hardware systems cannot queue such an interrupt.

11 • The mechanism for the specification of hardware registers and of interrupt vector locations is different from the mechanism for memory specifications on many architectures.

12 • A task entry specified as being an interrupt entry can be called by other tasks as a normal rendezvous. This fact can lead to erroneous programming situations and can also slow down interrupt code because it must take the general entry call into account.

13 • Entries need to be dynamically connected to an interrupt. For reasons of reconfigurability and fault tolerance, the configuration of an interrupt/entry pair often must change dynamically.

- Non-blocking interprocess communication is required so that a time-critical task can transmit parameters to other tasks without scheduler intervention. 14

In order to deal with these difficulties, implementations are resorting to various non-standard (and therefore non-portable) mechanisms, such as special pragmas. 15

6.4 Operations on Pointers

User Need 27 — Pointer Operations: In embedded applications, it is quite often the case that references to statically allocated objects at disjoint locations need to be obtained and kept as component values of composite objects, e.g., to form an array of references to these objects for efficient access by algorithms that iterate over the referenced objects, which may be located at pre-determined addresses. Objects obtained through allocators are insufficient for such purposes due to the need to influence their placement. 1

Requirement 27.1 — Pointers to Declared Objects: Ada 9X shall support pointers to objects created by declarations in addition to access values for objects created by allocators. Ada 9X shall provide operations for obtaining a pointer to an object having an appropriate pointer type; these operations shall maintain strong typing. Ada 9X shall also provide operations for converting machine addresses to an appropriate pointer type. 2

Discussion: The ability to have a flexible, general mechanism for binding objects to names at run time is fundamental to programming. For example, (1) when an object is created in a declarative part or package specification and the programmer would like to force the semantics of call-by-reference, or (2) when the objects of a type are forced to particular memory locations but the number and placement of these objects is not known until execution time, a mechanism is needed to allow access to the objects from a set of address values. 3

It is realized that use of the features described in this section is potentially unsafe. 4

7 Requirements for Parallel and Distributed Processing

The use of distributed multiple computers to implement large-scale resource-intensive applications has increased significantly since the adoption of Ada 83. In the future, this use is expected to proliferate as the advantages of distributed execution become economically attractive to more applications. Typically, distributed applications demand the reliability and safety afforded an Ada 83 main program combined with dynamic configuration facilities to optimize the use of available computing resources. 1

Currently, Ada 83 can be used to program applications for multiple computers. Unfortunately, the fact that practitioners are not able to do so with a unified semantic model has led to disparate approaches by individual applications. These approaches either restrict the unit of distribution to Ada main programs or introduce an artificial unit of distribution based upon the notion of a virtual node. In each approach, applications rely upon different degrees of extra-lingual support, thereby compromising the dynamic control of and type-safety among the program units comprising the application. Moreover, the lack of a unified semantic model leads to non-portability of applications. 2

Therefore, Ada 9X should specify a model of program execution that minimally extends the language to accommodate both the logical and physical separation of an application for distributed execution. This model must not preclude the continued use of existing approaches but should allow these approaches to transition gradually towards obsolescence. In addition, the model must not undermine the use of existing program units. 3

Applications for which the needs discussed in this chapter are essential include command and control systems, fault-tolerant systems, trusted systems, safety-critical systems, information systems, and computationally-intensive parallel systems. 4

7.1 Distribution of Single Programs

User Need 28 — Distribution of Single Programs: Ada 9X should provide standardized ways to write Ada programs that will execute on distributed systems. 1

Study Topic 28.1 — Distribution of Single Programs: An attempt shall be made in the design of Ada 9X to make it easier to distribute a single Ada program across a distributed architecture. Such enhancements might include, for example: 2

1. The specification of exact semantics (behavior and timing, including failure modes) of timed and conditional entry calls, potentially remote subprogram calls, and exception propagation. 3
2. The specification of exact semantics and available recovery from hardware failure. 4
3. Addressing the issue that a single package SYSTEM for the whole program implies certain commonality among the hardware processors executing the program. 5

4. Addressing the implication that there is a single simultaneous CLOCK across the whole system and a fixed granularity to DURATION.

Discussion: Distributed Ada systems will become more prevalent in the future. One form of software for a distributed system is a single program that is partitioned so that parts of it can be distributed to separate processing elements. An improved specification of the semantics of Ada that accounts more realistically for execution on a distributed architecture is needed. It is also necessary to remove implications from the language that the processing elements executing a distributed program share common hardware characteristics and a common notion of time.

A second form of distributed software system is to have multiple programs that run on separate processing elements, with perhaps more than one Ada program running on a given processor. The recommended enhancement for this type of system is to develop a standard interface for inter-program communication and synchronization. This standard interface need not be an integral part of the Ada 9X language. Hence, it has been recommended as an "external" standard, to be developed outside the Ada 9X mapping and revision effort.

7.2 Distribution of Ada Applications

User Need 29 — Distributing an Ada application: It must be possible to structure an Ada 9X application for execution using multiple separate computers. Doing so requires that an application have programmatic control over the aggregation of program units (unit of distribution) that must be executed on a single computer. The resulting application must retain the same degree of type-safety as for an Ada 83 main program.

Requirement 29.1 — Unit of Homogeneous Distribution: Ada 9X shall define a unit of distribution for structuring an application for distributed execution across separate homogeneous computers.

Study Topic 29.2 — Unit of Heterogeneous Distribution: Ada 9X shall define a unit of distribution for structuring an application for distributed execution across separate heterogeneous computers.

Discussion: Ada 83 does not provide a suitable program unit for distributing an application across separate computers. That the currently available program units are unsuitable is evident from the various alternative approaches that have been devised. These approaches usually impose restrictions that detract from the level of abstraction for representing the units of distribution. Therefore, a straightforward approach is required that maintains the proven legacy of reliability and safety of Ada 83 for non-distributed programming. This approach must not be detrimental or necessary to the programming of non-distributed applications.

It is recognized that programming an application distributed across heterogeneous computers presents a difficulty: a single package SYSTEM may be untenable. Therefore, this form of distribution is categorized as a study topic rather than a requirement.

7.3 Remote Communication

User Need 30 — Communication within a Distributed Application: It must be possible to program the exchange of data among units that comprise an application executing across remote computers. This data exchange must be performed with abstractions that are compatible with those available for an application executing on a single computer. Furthermore, the semantic model for data exchanges that require conditional or timed communication must remove the anomalies that currently exist in Ada 83. 1

Requirement 30.1 — Remote Communication: Ada 9X shall define abstractions and the associated semantic model for communication among program units comprising an application executing across remote computers. 2

Discussion: Ada 83 provides abstractions for serialized and concurrent data exchange among program units that are oriented towards non-distributed execution. Adapting these abstractions for an application that is distributed across remote computers requires that the model of communication be enhanced to specify failure semantics and to resolve synchronization and timing anomalies. 3

7.4 Configuration of Distributed Systems

User Need 31 — Configuring an Ada Application: It must be possible to control the configuration of computers upon which an Ada application executes. This control includes programming both the initial configuration and subsequent configurations so as to optimize the use of available computing resources. Different configurations must remain transparent to the application; that is, a program unit must execute independently of the computer on which it resides. 1

Requirement 31.1 — Unit of Configuration: Ada 9X shall define a unit of configuration for controlling the distributed execution of an application across multiple computers. 2

Discussion: Ada 83 does not provide sufficient programmatic control of the computers upon which an application executes. For example, applications must rely upon extra-lingual support to achieve fault-tolerant execution in the event of computer failure. Typically, this support compromises the application to the extent that "alien" abstractions intrude into the programming. Consequently, an application must be programmed without the benefit of a unified linguistic model that retains the proven reliability and safety of Ada 83. 3

8 Requirements for Safety-Critical and Trusted Applications

Safety-critical applications, such as commercial avionics, medical equipment, and others, often have special requirements arising from the need to certify the code.

8.1 Predictability of Execution

User Need 32 — Reduce the Complexity of Implementation Dependence: Ada 83 permits implementors many areas in which they make choices. It is necessary to reduce the complexity of implementation dependence in Ada 9X.

Requirement 32.1 — Minimize Implementation Dependencies: The discussion in Section 2.4 on page 11 covers this requirement.

User Need 33 — Generated Code Analyzability: Users who certify the correctness of safety critical software typically compare source and generated code to ensure that the generated code is correct. Optimizations introduced by the compiler may obscure the relationship between source and generated code and make this examination excessively difficult. These users require support that reduces both the tedium and the amount of work needed for this comparison. In the past this requirement has led to the use of compilers that translate source code in a very literal manner without introducing any subtle optimizations. This practice has simplified source and generated code examinations. For Ada 9X simply suppressing the compiler's optimization process may not be adequate if this suppression does not make it easier to compare source and generated code.

Requirement 33.1 — Retain Correspondence Between Source Code and Object Code: A mode of operation of Ada 9X compilers is required that, to the extent practical, retains the correspondence between source code and object code.

Study Topic 33.2 — Literal Translation Mode: The Ada 9X standard shall allow implementations that generate code in a manner that simplifies the analysis needed for program certification. These implementations may use techniques such as the suppression of compiler's optimization process. They may reject programs that they cannot compile in this manner.

Discussion: The safety critical and trusted systems users of Ada 9X require control of the code generation performed by the compiler. Specifically, they require a means to cause the compiler to generate code that a human can analyze. The purpose of the analysis is usually to demonstrate, and sometimes prove, that the source code and the generated code have the same meaning.

The selection of this form of compilation must be shown in the source code.

Users have described this requirement as a need for optimization suppression, since optimizations have sometimes complicated the analysis of the generated code. However, some optimizations may simplify analysis, and these would then be desirable. Also, given the complex nature of the full Ada language, there may be source programs that the compiler cannot compile in an easily

analyzed form. Programs containing tasks, interfaces to other languages such as SQL, or floating point arithmetic are examples. The compiler should be permitted to indicate these situations with warning or error messages.

- This requirement does not demand new language features. Instead, it indicates a need to understand the restrictive kind of programming used in this domain, and a need to allow Ada compilers to support this programming style.

10 **Requirement 33.3 — Document Uses of Support Code:** Implementors must document all cases in which code is executed other than that which the programmer writes (that is, code in support packages). Moreover, it must be practical to program in an Ada subset with the property that no such code is executed.

9 Requirements for Information Systems

Information systems typically require the capability to deal with character data in a flexible manner and with large currency amounts. Many Information Systems applications use non-ASCII character sets, in particular EBCDIC, and these alternate character sets must be fully supported in Ada without loss of efficiency. The requirement for character handling can be met with suitable secondary standards, but the requirements for currency manipulation and alternate character support require language modifications.

9.1 Handling of Currency Quantities for Information Systems

User Need 34 — Support for Currency Quantities: Information Systems programs that deal with quantities representing large currency amounts need to be able to perform exact arithmetic on these quantities, with control over rounding. Precisions as large as those supported by COBOL are required.

Requirement 34.1 — Decimal-Based Types: Ada 9X shall provide types capable of supporting at least the level of arithmetic capability found in COBOL, including 18 digits of decimal precision.

Discussion: The current fixed-point model in Ada does not meet this requirement in a number of respects:

- It is currently unclear, in practice, whether implementations must support non-binary values of *small*, and in particular the issue of mixing types with incompatible *small* values has raised implementation issues that have limited the support for decimal values of *small*.
- There are a number of problems with the *small* representation clause, including the placement rules and the fact that it is not really a representation clause at all since it affects the semantics.
- Ada fixed point is oriented to its use as a replacement for floating point, when exact results are not required. Consequently the model for fixed-point is close to that of floating-point, whereas the arithmetic facility required for information systems is closer to scaled integer arithmetic. In particular, neither stored values nor intermediate results should be permitted to be stored with extra precision in this context. Extra range is of course permitted (actually required) for intermediate results.
- Deterministic semantics with control over rounding are required in the currency computation case, as opposed to model number non-determinism. Extra precision, even if restricted only to intermediate values, can cause unexpected results.

Moreover, for reasons of efficiency and I/O compatibility it appears to be impractical to write a package that exports the requisite types.

Study Topic 34.2 — Specification of Decimal Representation: Ada 9X information system implementations shall provide a mechanism for specifying decimal representations for decimal scaled data.

10 **Discussion:** Information Systems applications typically make heavy use of decimal representations (both 8-bit character and 4-bit packed decimal) for decimal scaled data. There are three reasons for supporting this feature in Ada:

11 • On many architectures used for Information Systems applications, the instruction set provides direct support for arithmetic using these representations, and Ada 9X should not preclude making use of this hardware.

12 • It commonly arises in this type of application that quantities are manipulated in contexts requiring external representation (for example, input/output) much more frequently than in computational contexts, so decimal representations can be more efficient even in the absence of hardware support.

13 • Interfacing to COBOL and to other related tools, including most data base managers, and to files generated by COBOL programs, requires support for decimal representations.

14 The issue of whether representations of this type should be completely general, and usable throughout a program, or more restricted (concentrating on the interface requirements) needs to be evaluated.

9.2 Compatibility with Other Character Sets

1 **User Need 35 — Alternate Character Set Support:** Many Information Systems applications are required to interface smoothly with applications based on character sets other than ASCII, particularly EBCDIC.

2 **Study Topic 35.1 — Alternate Character Set Support:** Ada 9X information-system implementations shall provide for the extension of the set of graphic symbols to obtain input/output facilities and 'IMAGE attributes for alternate character sets (specifically, for EBCDIC) comparable in both functionality and performance with the features provided for the built-in character set.

3 Note also the international requirements described in Section 3.1 on page 13.

9.3 Interfacing with Data Base Systems

1 Often programs written in Ada must interface with Data Base Management Systems (DBMSs) which were designed to interface with COBOL programs.

2 **User Need 36 — Interfacing Ada Programs to DBMSs:** It should be relatively easy to write Ada 9X programs that interface smoothly with DBMSs designed with COBOL programs.

9.4 Common Data Structures

User Need 37 — Standard Data-Manipulating Features: Ada 9X should provide, perhaps as secondary standards, a collection of modules for dealing with various standard kinds of data, such as varying length strings, stacks, queues, and such.

10 Requirements for Scientific and Mathematical Applications

10.1 Floating Point

User Need 38 — Floating Point Facilities: There is a need for more predictability in the results of floating-point operations on any given floating-point architecture. There is also a need for uniform access to common mathematical functions. 1

Study Topic 38.1 — Floating-Point Facilities: Ada 9X shall require that the results of floating-point operations be predictable from the documentation provided for any given implementation. It should provide a framework for such descriptions that promotes uniformity of results. It shall allow implementations that fully support the IEEE-754 [7] floating-point standard. It shall provide standard interfaces to elementary mathematical functions. 2

Discussion: The Ada 83 formal model for floating-point uses non-determinism to conform to almost all floating-point architectures. Subsequent experience indicates that it is desirable for results to be deterministic and well-specified on any given architecture, with the language providing a framework for, and constraints on, these specifications. The proposed ISO Language Compatible Arithmetic Standard (LCAS) is an example of such a framework. 3

Ada 83 makes it difficult to accommodate certain features of the IEEE-754 standard, such as output of -0.0 and input of infinities. There are few such constraints, and they should be easy to relax. 4

Ada 83 does not standardize on mathematical functions that are commonly provided in other languages — notably square root, trigonometric functions, logarithm, and exponentiation. Incorporation of the proposed Generic Elementary Function package, for example, will satisfy this requirement. 5

10.2 Data Storage

The Ada 83 standard does not specify the way array components are laid out in memory. The following points are of interest: 1

- For increasing index, it is necessary that the values of the addresses where the components are stored change monotonically. 2
- For multi-dimensional array objects, there are three storage modes of interest: row major, column major, or other: 3
  - Row major: For multi-dimensional arrays, the last index runs the fastest, then the last but one, etc. 4
  - Column major: For multi-dimensional arrays, the first index runs the fastest, then the next, etc. 5
  - Other storage methods include sparse matrices, or block storage. 6

This information is needed because many linear algebra algorithms have both row-oriented and column-oriented variants, and the efficiency of these variants depends strongly on the storage mode used. 7

- In general, where there is an efficiency difference in the way indexing is implemented, compilers should do it in the most efficient way.
- For applications which communicate with programs written in other languages (often Fortran), it is necessary to direct the Ada compiler to match the requirements of that language.

10 **User Need 39 — Data Layout in Memory:** Programmers of numerically intensive computing applications need both to be able to determine how the compiler chooses to allocate storage and to control that layout.

11 **Study Topic 39.1 — Data Layout in Memory:** The Mapping/Revision Team shall develop suitable attributes to permit the programmer to interrogate the way array storage is laid out.

12 Additionally, the Ada 83 user has no (implementation-independent) option to choose a preferred storage mode. It can be imagined that some Ada implementations would allow both row major and column major matrix storage and leave the choice to the user. We assume that in such cases it is not feasible to have the possibility of choosing the storage mode for *every* individual matrix object, but rather for all multi-dimensional array objects collectively. Then, if all matrices are stored the same way dead-code elimination would be possible. We do not want to require this additional facility, but only wish to mention it here as a related issue.

References

[1] *Ada 9X Project Plan*
Office of the Under Secretary of Defense for Acquisition, Washington, D.C. 20301, January 1989.

[2] *Ada Board's Recommended Ada 9X Strategy*
Office of the Under Secretary of Defense for Acquisition, Washington, D.C. 20301, September 1988.

[3] Real-Time Tasking Semantics Working Group.
Asynchronous Communication Between Tasks.
Ada Letters X(4):35-39, Spring, 1990.
Proceedings of the Third International Workshop on Real-Time Ada Issues.

[4] Baker, T.
Fixing Some Time-Related Problems in Ada.
Ada Letters X(4):136-143, Spring, 1990.
Proceedings of the Third International Workshop on Real-Time Ada Issues.

[5] Baker, T.
Timing Issues Working Group.
Ada Letters X(4):119-135, Spring, 1990.
Proceedings of the Third International Workshop on Real-Time Ada Issues.

[6] Brender, R. F.
Character Set Issues for Ada 9X.
Technical Report, Software Engineering Institute, October, 1989.

[7] IEEE.
IEEE Standard for Floating Point Arithmetic, ANSI/IEEE Standard 754-1985
1985.

[8] ISO 646: 7-Bit Coded Character Set.

[9] ISO 8859: 8-bit Single-Byte Coded Character Sets.
1987.

[10] Real-Time Tasking Semantics Working Group.
Real-Time Performance Standard.
Ada Letters X(4):40-45, Spring, 1990.
Proceedings of the Third International Workshop on Real-Time Ada Issues.

[11] Quiggle, T. J.
Ramifications of Re-introducing Asynchronous Exceptions.
Ada Letters X(4):25-31, Spring, 1990.
Proceedings of the Third International Workshop on Real-Time Ada Issues.

Appendix A: Simplicity Requirements

A.1 Understandability

1 In Section 2.2, User Need 5 on page 9 calls for improving the understandability of Ada 9X. Following is a list of possible areas for improvement.

elaboration order

2 Ada 83 rules for elaboration order often permit entities to be accessed before their declarations are elaborated and before the objects are fully initialized. Ada 83 provides an explicit mechanism — the ELABORATE pragma — to allow programmers to specify additional constraints on the elaboration order of library units and unit bodies. This pragma allows one to specify that elaboration of a particular unit's body must precede elaboration of the compilation unit containing the pragma. Unfortunately, this mechanism has drawbacks, the principal one being that the pragma does not transitively force elaboration of units named by its target's context clauses. Its use therefore frequently requires a client unit to know the dependencies of that target, violating an important abstraction.

3 Ada 9X shall attempt to provide mechanisms allowing programmers explicit control over the order in which library units and unit bodies are elaborated. The rules for (automatic) elaboration of library units and unit bodies shall be defined so as to minimize the instances in which these explicit mechanisms must be used. As much as possible, these rules should avoid requiring clients of a library unit to know details of the implementation and the dependencies of the library unit or its body.

later declarative items

4 Unnecessary difficulties arise because Ada 83 distinguishes between basic declarative items and later declarative items (see ARM Section 3.9). The distinction should be removed.

5 required **return** from a function

Ada 83's requirement (6.5(1)) of at least one **return** statement in a function body is an unnecessary annoyance during development, when the body might either be empty or consist of merely a **raise** statement.

6 visibility of literals and operations

Ada 83 provides facilities to control visibility of declarations among packages. However, the rules do not provide visibility of literals and operations in certain situations where the type itself is visible. Of the two ways of establishing direct visibility in Ada 83, use-clauses may allow much more than just literals and desired operations to become visible, and renaming can be cumbersome and does not allow character literals to be renamed.

7 Ada 9X shall provide a mechanism to establish direct visibility of the literals for enumeration and character types, without importing all of the declarations in the scope where the type is declared and without explicitly mentioning each imported literal.

8 Ada 9X shall provide a mechanism to establish direct visibility of the infix and prefix operations of a type without importing all declarations in the scope where the type is declared and without explicitly mentioning each imported operation.

access before elaboration

9 A consequence of Ada's unique concept of elaboration is that elaboration is required for entities that do not seem to need it. Examples where this effect occurs include:

constant objects initialized with compile-time-computable expressions, subprograms with no non-local references, and packages that (transitively) contain only such entities. The required elaboration causes complications for programmers attempting to use Ada program units where there is no Ada main program. (This problem is sometimes referred to as "call-in" in contrast to "call-out" which is provided by pragma INTERFACE.) Another manifestation of this problem is that one cannot declare a type T, an object of type T, and a T-valued function in the visible part of a package and use the function to initialize the object in its declaration.

The Mapping/Revision Team should attempt to modify the Ada elaboration rules to facilitate "call-in" of Ada subprograms from non-Ada code (as when Ada is used to implement subprogram libraries) and to address implementation issues raised by the access-before-elaboration check. 10

Ada 9X should require implementors to specify those conditions under which real-time implementations will not perform run-time elaboration of constructs. In particular, these conditions will specify constructs that need no elaboration because their characteristics are constant and known before execution. 11

A.2 Generality

In Section 2.2 on page 8, User Need 6 on page 9 calls for minimizing special-case restrictions in Ada 9X. Following is a list of possible areas for improvement. 1

out parameters to functions

Procedure subprograms may have **out** parameters; function subprograms may not. Ada 9X should remove this distinction as it serves to inhibit only one form of function side effects while not limiting others. 2

reading out parameters

An **out** parameter may be written to within a subprogram body but not read; remove this restriction. 3

ability to redefine "="

Ada 83 programmers are able to redefine equality using a subterfuge involving generics. Ada 9X should be modified to permit doing so directly. 4

flexibility in allowed optimization

While Ada 9X users need compilers that employ the best available techniques for code optimization, they also need facilities to control the optimization process. In some situations they must be able to suppress it completely within a region of a program; in others they need to be able to direct how trade-offs will be made. Some users will tolerate minor execution side effects if these lead to major throughput improvements. The user must be able to express these intentions in the source code. 5

The Ada 9X standard shall permit the use of flexible and powerful code optimization techniques, especially those that permit the target hardware to be fully utilized. The programmer shall have a means to direct the compiler's optimizations with elements included in the source code. These controls shall include optimization suppression, and may include other options in addition to those now available. 6

Code optimizations performed by a compiler can dramatically improve a program's execution characteristics. For example, increasing a program's size can also increase its speed in some cases. There are many factors involved in these trade-off decisions, and Ada 9X users need facilities to express a wide range of options. They particularly need a means to suppress all optimizations. However, they also want to be able to 7

receive the full benefit of vector processing, common subexpression elimination, and reordering optimizations applied to larger regions of the program. The compiler should be able to apply optimizations that include inlined subprograms and to reorder assignment statements. The Ada 83 standard, in Section 11.6, states overly restrictive optimization constraints. Increased flexibility, accompanied with increased user control, would be beneficial.

- Note the conflict between this requirement and Requirement 33.1 on page 37.

anonymous array types

- Ada 83 often requires array type declarations which define types that are used only once. Ada 9X should permit anonymous array types in more contexts, such as: